

Citation for published version:

Shams, Z, De Vos, M, Padget, J & Vasconcelos, W 2017, 'Practical Reasoning with Norms for Autonomous Software Agents', *Engineering Applications of Artificial Intelligence*, vol. 65, pp. 388-399.
<https://doi.org/10.1016/j.engappai.2017.07.021>

DOI:

[10.1016/j.engappai.2017.07.021](https://doi.org/10.1016/j.engappai.2017.07.021)

Publication date:

2017

Document Version

Peer reviewed version

[Link to publication](#)

University of Bath

Alternative formats

If you require this document in an alternative format, please contact:
openaccess@bath.ac.uk

General rights

Copyright and moral rights for the publications made accessible in the public portal are retained by the authors and/or other copyright owners and it is a condition of accessing publications that users recognise and abide by the legal requirements associated with these rights.

Take down policy

If you believe that this document breaches copyright please contact us providing details, and we will remove access to the work immediately and investigate your claim.

Practical Reasoning with Norms for Autonomous Software Agents

Zohreh Shams^{a,1,*}, Marina De Vos^a, Julian Padget^a,
Wamberto W. Vasconcelos^b

^a*Department of Computer Science, University of Bath, UK (BA2 6AH)*

^b*Department of Computing Science, University of Aberdeen, UK (AB24 3UE)*

Abstract

Autonomous software agents operating in dynamic environments need to constantly reason about actions in pursuit of their goals, while taking into consideration norms which might be imposed on those actions. Normative practical reasoning supports agents making decisions about what is best for them to (not) do in a given situation. What makes practical reasoning challenging is the interplay between goals that agents are pursuing and the norms that the agents are trying to uphold. We offer a formalisation to allow agents to plan for multiple goals and norms in the presence of *durative* actions that can be executed *concurrently*. We compare plans based on decision-theoretic notions (i.e. utility) such that the utility gain of goals and utility loss of norm violations are the basis for this comparison. The set of *optimal* plans consists of plans that maximise the overall utility, each of which can be chosen by the agent to execute. We provide an implementation of our proposal in Answer Set Programming, thus allowing us to state the original problem in terms of a logic program that can be queried for solutions with specific properties.

Keywords: Intelligent Agents, Practical Reasoning, Norms, Goals

*Corresponding author.

Email addresses: `z.shams@bath.ac.uk` (Zohreh Shams), `m.d.vos@bath.ac.uk` (Marina De Vos), `j.a.padget@bath.ac.uk` (Julian Padget), `wasconcelos@acm.org` (Wamberto W. Vasconcelos)

¹Present Address: Computer Laboratory, University of Cambridge, UK (CB3 0FD)

1. Introduction

Reasoning about what to do – known as practical reasoning – for an agent pursuing different goals is a complicated task. When conducting practical reasoning, the agents might exhibit undesirable behaviour that was not predicted. The necessity of controlling undesirable behaviour has given rise to the concept of norms that offer a way to define ideal behaviour for autonomous software agents in open environments. Such norms often define obligations and prohibitions that express what the agent is obliged to do and what the agent is prohibited from doing.

Depending on their computational interpretation, norms can be regarded as *soft* or *hard constraints*. When modelled as hard constraints, the agent subject to the norms is said to be *regimented*, in which case the agent has no choice but to blindly follow the norms (Esteva et al., 2001). Although regimentation guarantees norm compliance, it greatly restricts agent autonomy. Moreover, having individual goals to pursue, self-interested agents might not want to or might not be able to comply with the norms imposed on them. Conversely, *enforcement* approaches, in which norms are modelled as soft constraints, leave the choice of complying with or violating the norms to the agent. However, in order to encourage norm compliance, there are consequences associated, namely a punishment when agents violate a norm (López y López et al., 2005; Pitt et al., 2013) or a reward when agents comply with a norm (Aldewereld et al., 2006). In some approaches (*e.g.*, Aldewereld et al. (2006); Alrawagfeh and Meneguzzi (2014); Oren et al. (2011)) there is a utility gain/loss associated with respecting norm or not, whereas in the *pressured norm compliance* approaches (*e.g.*, López y López et al. (2005)), the choice to violate a norm or not is determined by how the norm affects the satisfaction or hindrance of the agent’s goals.

Existing work (*e.g.* Oren et al. (2011); Panagiotidi et al. (2012a); Criado et al. (2010); Meneguzzi et al. (2015)) on normative practical reasoning using enforcement either consider plan generation or plan selection where there is a set of pre-generated plans available to the agent. In these works, the attitude agents have toward norms is often one of compliance, meaning that their plans are often selected or, in some approaches, customised, to ensure norm compliance (*e.g.*, Kollingbaum (2005); Alechina et al. (2012); Oren et al. (2011)). We argue that in certain situations, an agent might be better off violating a norm which, if followed, would make it impossible for the agent to achieve an important goal or complying with a more important norm.

In this paper we set out an approach for practical reasoning that considers norms in both plan generation and plan selection. We extend current work on normative plan generation such that the agent attempts to satisfy a set of potentially conflicting goals in the presence of norms, as opposed to conventional planning problems that generate plans for a single goal (Oren et al., 2011; Panagiotidi et al., 2012a). Such an extension is made on top of STRIPS (Fikes and Nilsson, 1971), the most established planning domain language that lays the foundation of many automated planning languages. Additionally, since in reality the actions are often non-atomic, our model allows for planning with durative actions that can be executed concurrently. Through our practical reasoning process agents consider *all* plans (i.e., sequences of actions), including those leading to norm compliance and violation; each plan gets an associated overall utility for its sequence of actions, goals satisfied, and norms followed/violated, and agents can decide which of them to pursue by comparing the relative importance of goals and norms via their utilities. The plan an agent chooses to follow is not necessarily norm-compliant; however, our mechanism guarantees that the decision will maximise the overall plan utility, and this justifies the occasional violation of norms in a plan. Both plan generation and plan selection mechanisms proposed in this paper are implemented using Answer Set Programming (ASP) (Gelfond and Lifschitz, 1988).

ASP is a declarative programming paradigm using logic programs under Answer Set semantics. In this paradigm the user provides a description of a problem and ASP works out how to solve the problem by returning answer sets corresponding to problem solutions. The existence of efficient solvers to generate the answers to the problems provided has increased the use of ASP in different domains of autonomous agents and multi-agent systems such as planning (Lifschitz, 2002) and normative reasoning (Cliffe et al., 2006; Panagiotidi et al., 2012b). Several action and planning languages such as event calculus (Kowalski and Sergot, 1986), \mathcal{A} (and its descendants \mathcal{B} and \mathcal{C} (Gelfond and Lifschitz, 1998), Temporal Action Logics (TAL) (Doherty et al., 1998), have been implemented in ASP (Lee and Palla, 2012, 2014), indicating that ASP is appropriate for reasoning about actions. This provides motive and justification for an implementation of STRIPS (Fikes and Nilsson, 1971) that serves as the foundation of our model in ASP.

This paper is organised as follows. First we present a scenario in Section 2 which we use to illustrate the applicability of our approach. This is followed by the formal model and its semantics in Section 3. The computational

implementation of the model is provided in Section 4. After the discussion of related work in Section 5, we conclude in Section 6.

2. Illustrative Scenario

To illustrate our approach and motivate the normative practical reasoning model in the next section, we consider a scenario in which a software agent acts as a supervisory system in a disaster recovery mission and supports human decision-making in response to an emergency. The software agent’s responsibility is to provide humans with different courses of actions available and to help humans decide on which course of actions to follow. In our scenario, the agent is to plan for a group of human actors who are in charge of responding to an emergency caused by an earthquake. The agent monitors the current situation (e.g., contamination of water, detection of shocks, etc.) and devises potential plans to satisfy goals set by human actors. In our scenario we assume the following two goals:

1. Running a hospital to help wounded people: this goal is fulfilled when medics are present to offer help and they have access to water and medicines.
2. Organising a survivors’ camp: this is fulfilled when the camp’s area is secured and a shelter is built.

We also assume the two following norms that the agent has to consider while devising plans to satisfy the goals above:

1. It is forbidden to build a shelter within 3 time units of detecting shocks. The cost of violating this norm is 5 units.
2. It is obligatory to stop water distribution for 2 time units once poison is detected in the water. The cost of violating this norm is 10 units.

The formulation of this scenario is provided in Appendix A.

3. A Model for Normative Practical Reasoning

We use STRIPS (Fikes and Nilsson, 1971) as the basis of our normative practical reasoning model. In STRIPS, a planning problem is defined in terms of an initial state, a goal state and a set of operators (e.g. actions). Each operator has a set of preconditions representing the circumstances/context in which the operator can be executed, and a set of postconditions that result from applying the operator. Any sequence of actions that satisfies the goal is a solution to the planning problem. In order to capture the features of

the normative practical reasoning problem that we are going to model, we extend the classical planning problem by:

- (i) replacing atomic actions with *durative* actions: often the *nature* of the actions is non-atomic, which means that although executed atomically in a state, the system state in which they finish executing is not necessarily the same in which they started (Nunes et al., 1997). Refinement of atomic actions to durative actions reflects the real time that a machine takes to execute certain actions.
- (ii) Allowing a set of potentially inconsistent goals instead of the conventional single goal: the issue of planning for multiple goals distributed among distinct agents is addressed in collaborative planning. We address this issue for a single agent.
- (iii) Factoring in norms: having made a case for the importance of norms in Section 1, we combine normative and practical reasoning. Just like goals, a set of norms is not necessarily consistent, making it potentially impossible for an agent to comply with all norms imposed on it.

A solution for a normative practical reasoning problem that features (i), (ii) and (iii) is any sequence of actions that satisfies at least one goal. The agent has the choice of violating or complying with norms triggered by execution of a sequence of actions, while satisfying its goals. However, there may be consequences either way that the agent has to foresee.

We explain the syntax and semantics of the model in Sections 3.2–3.6. First we present the architecture of our envisaged system in the next section.

3.1. Architecture

The architecture, depicted in Figure 1, shows how re-planning can be considered if a plan in progress is interrupted due to a change in circumstances. This change can be the result of a change in the environment or unexpected actions of other agents in the system. As is customary in multi-agent systems, the agent will regularly check the viability of its plan. The frequency depends on the type of system the agent is operating in, the agent’s commitment to its intentions, and the impact of re-computation on the agent’s overall performance.

When an agent decides that re-planning is in order, it will take the state in which the plan is interrupted as the initial state for the new plan and its current goal set as the goals to plan towards. The current goal set does not have to be the same as the goal set the original plan was devised for. Goals can already be achieved in the interrupted plan, previous goals may no longer

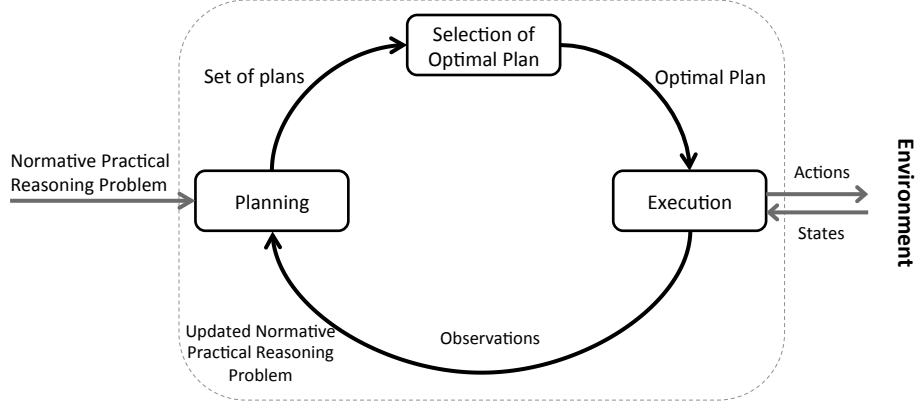


Figure 1: Overview of the System Architecture

be relevant and others may have been added. Even if the goals remain the same, the resulting new optimal plan might change due to changes in the state of the system. Similarly, there might be new norms imposed on the agent that will have to be considered in replanning.

We cater for agents which need to create their own individual plans. However, in doing so, in multi-agent scenarios agents will inevitably interact and interfere with each other’s plans. The architecture in Figure 1 will cater for this in two ways: (i) agents will notice the “States” being changed by other agents (a way of indirect communication) and (ii) the “Observations” will contain interactions among agents (direct communication).

3.2. Syntax

We start by describing an extended STRIPS planning problem, defined in (Shams, 2016), that accommodates (i) durative actions; (ii) multiple goals and (iii) multiple norms.

Definition 1 (Normative Practical Reasoning Problem). *A normative practical reasoning problem is a tuple $P = (FL, \Delta, A, G, N)$ where*

- (i) FL is a set of fluents;
- (ii) Δ is the initial state;
- (iii) A is a finite, non-empty set of durative actions for the agent;
- (iv) G is the agent’s set of goals;
- (v) N is a set of norms.

We describe in the ensuing sections each of the components of a normative practical reasoning problem.

3.2.1. Fluents and Initial State

FL is a set of domain fluents describing the domain the agent operates in. A literal l is a fluent or its negation i.e. $l = fl$ or $l = \neg fl$ for some $fl \in FL$. For a set of literals L , we define $L^+ = \{fl \mid fl \in L\}$ and $L^- = \{fl \mid \neg fl \in L\}$ to denote the set of positive and negative fluents in L respectively. L is well-defined if there exists no fluent $fl \in FL$ such that $fl, \neg fl \in L$, i.e., if $L^+ \cap L^- = \emptyset$.

The semantics of the normative practical reasoning problem is defined over a set of states Σ . A state $s \subseteq FL$ is determined by set of fluents that hold *true* at a given time, while the other fluents (those that are not present) are considered to be false. A state $s \in \Sigma$ satisfies fluent $fl \in FL$, denoted $s \models fl$, if $fl \in s$. It satisfies its negation $s \models \neg fl$ if $fl \notin s$. This notation can be extended to a set of literals as follows: set X is satisfied in state s , $s \models X$, when $\forall x \in X, s \models x$.

The set of fluents that hold at the initial state is denoted by $\Delta \subseteq FL$.

3.3. Durative Actions

The component A of our normative practical reasoning problem $P = (FL, \Delta, A, G, N)$ is a set of durative actions. A durative action has pre- and post-conditions. The effects of an action (as captured by its post-conditions) are not immediate, that is, it takes a non-zero period of time for the effects of an action to take place.

Definition 2 (Durative Actions). *A durative action a is a tuple $\langle pr, ps, d \rangle$ where pr and ps are possibly empty and finite sets of well-defined literals representing respectively the pre- and post-conditions of the action, and $d \in \mathbb{N}, d > 0$, is the duration of the action.*

Given an action $a = \langle pr, ps, d \rangle$ we may also refer to its three components $pr(a)$, $ps(a)$ and d_a . Moreover, we use $pr(a)^+$ and $pr(a)^-$ to refer to, respectively, the positive and negative literals in $pr(a)$; similarly, we have $ps(a)^+$ and $ps(a)^-$ to refer to respectively the positive and negative literals in $ps(a)$.

An action a can be executed in a state s if its preconditions hold in that state (i.e. $s \models pr(a)$). When modelling durative actions, there might be several states between the start and end state of the action, during which the action is said to be *in progress*. We assume that the preconditions of actions have to be preserved during their execution. The postconditions of a durative action cause changes to the state s in which the action ends. These

changes are: adding the positive postconditions $ps(a)^+$ to s and deleting the negative postconditions $ps(a)^-$ from s . Thus, for a state s in which action a ends, we have: $s \models ps(a)^+$ and $s \not\models ps(a)^-$.

Example 1. *To build a shelter (action “buildShelter”) the agent has to secure and evacuate the area and there has to be no shock detected – these are represented as preconditions $areaSecured$, $evacuated$ and $\neg shockDetected$, respectively. Once the shelter is built the area does not have to remain evacuated. This is represented as the positive postcondition $shelterBuilt$, while the negative postcondition of this action is $\neg evacuated$. In our scenario, we model this action as taking 4 units of time (that is, if it is executed in state s_k , it will end in state s_{k+4}).*

$$buildShelter = \left\langle \left\{ \begin{array}{c} areaSecured, \\ evacuated, \\ \neg ShockDetected \end{array} \right\}, \left\{ \begin{array}{c} shelterBuilt, \\ \neg evacuated \end{array} \right\}, 4 \right\rangle$$

3.4. Goals

Goals identify the state of affairs in the world that an agent wants to achieve. We define below the elements of the set G of $P = (FL, \Delta, A, G, N)$.

Definition 3 (Goals). *A goal $g \in G$ is the pair $\langle r, v \rangle$, where r is a possibly empty and finite set of well-defined literals representing the goal requirements, and $v \in \mathbb{N}, v > 0$, represents the utility/gain for achieving the goal.*

Goal g ’s requirements and value are denoted as $r(g)$ and $v(g)$, respectively.

Example 2. *The goals from our illustrative scenario are formulated as below:*

$$runningHospital = \left\langle \left\{ \begin{array}{c} medicsPresent, \\ waterSupplied, \\ medicinesSupplied \end{array} \right\}, 25 \right\rangle$$

$$organiseSurvivorCamp = \left\langle \left\{ \begin{array}{c} areaSecured, \\ shelterBuilt \end{array} \right\}, 18 \right\rangle$$

3.5. Norms

In this section we specify what we refer to as a norm in this work. In order to provide a context for the norm specification we explain how our norm specification corresponds to the five elements identified by Pacheco (2012) that distinguish norm specification languages.

1. **Deontic Operators:** We model a permissive society in which the agent has complete knowledge of the domain of actions available. Everything is permitted unless it is explicitly prohibited. The role of obligation is to motivate the agent to execute a specific action and the role of prohibition is to inhibit the agent from executing a particular action.
2. **Controls:** Controls determine whether the deontic propositions operate on actions, states or both. In this work we focus on action-based norms.
3. **Enforcement Mechanisms:** We use the enforcement mechanism proposed by Shams et al. (2015) that is a combination of utility-based (e.g., Oren et al. (2011); Panagiotidi et al. (2012a)) and pressure-based (López y López et al., 2005) compliance methods.
4. **Conditional Expressions:** Similar to the control element, we use actions as conditional expressions. In other words, the norm condition is an action that once executed, the agent is obliged to or prohibited from executing the action that the norm targets.
5. **Temporal Constraints:** temporal constraints can be used to express norm activation, termination, deadline, etc. The temporal constraint we specify here is concerned with the deadline. The agent is expected to comply with an obligation (execute a certain action) or a prohibition (refrain from executing a specific action) before some deadline.

We now define the element N of problem $P = (FL, \Delta, A, G, N)$ that denotes a set of conditional norms to which the agent is subject:

Definition 4 (Norms). *N is a set of norms, each of which is a tuple of the form $n = \langle d_o, a_{con}, a_{sub}, dl, c \rangle$, where*

- $d_o \in \{o, f\}^2$ *is the deontic operator determining the type of norm, which can be an obligation or a prohibition;*
- $a_{con} \in A$ *is the durative action (cf. Def. 2) that activates the norm;*
- $a_{sub} \in A$ *is the durative action (cf. Def. 2) that is the target of the obligation or prohibition;*
- $dl \in \mathbb{N}$ *is the norm deadline relative to the activation condition, which is the completion of the execution of the action a_{con} ; and*
- $c \in \mathbb{N}$ *is the penalty cost that will be applied if the norm is violated. $c(n)$ denotes the penalty cost of norm n .*

²Symbols o and f are normally represented as **O** and **F** in the Deontic logic. However we have used lower case letters to make these consistent with our implementation in the next section. Capital letters in the implementation language are reserved for variables.

An obligation norm states that executing action a_{con} obliges the agent to start/start and end the execution of a_{sub} within dl time units of the end of execution of a_{con} . Such an obligation is complied with if the agent starts or starts and ends executing a_{sub} before the deadline and is violated otherwise. A prohibition norm expresses that executing action a_{con} prohibits the agent from starting or starting and ending the execution of a_{sub} within dl time units of the end of execution of a_{con} . Such a prohibition is complied with if the agent does not start or does not start and end executing a_{sub} before the deadline and is violated otherwise.

Example 3. *The norms from the illustrative scenario are:*

$$\begin{aligned} n_1 &= \langle f, detectShock, buildShelter, 3, 5 \rangle \\ n_2 &= \langle o, detectPoison, stopWater, 2, 10 \rangle \end{aligned}$$

A norm can be activated multiple times in a sequence of action, generating different instances of the original norm. To make sure different instances are dealt with uniquely, we define *instantiated norms*. In each instance the deadline is updated relative to the end of execution of the action that is the condition of the norm.

Definition 5 (Instantiated Norm). *An instantiation of norm $n = \langle d_o, a_{con}, a_{sub}, dl, c \rangle$ is denoted as $n_{ins} = \langle d_o, a_{con}, a_{sub}, dl_{ins}, c \rangle$ where $dl_{ins} = dl + t_{acon} + d_{acon}$. t_{acon} is when action a_{con} is executed and d_{acon} is the duration of a_{con} .*

We also denote an instantiation of a norm n_i as n'_i .

Example 4. *Assume that in some sequence of action $detectShock$ is executed at time 3 (i.e. $t_{acon} = 3$) and that the duration of this action is 1 (i.e. $d_{acon} = 1$). The instantiated version of norm $n_1 = \langle f, detectShock, buildShelter, 3, 5 \rangle$, in this sequence of actions is $n'_1 = \langle f, detectShock, buildShelter, 7, 5 \rangle$ Where dl_{ins} is calculated based on Def. 5.*

3.6. Semantics

Having explained the syntax of the model, we now focus on the semantics. To this end, we need to describe:

- (i) What the possible courses of action for the agent are and what properties each course of action has. Properties are defined in terms of goals that a sequence of action satisfies, norms it complies with and the norms it violates. This item is discussed in Section 3.6.1.

- (ii) What identifies a sequence of actions as a solution/plan for problem P .
Plans are defined in Section 3.6.2.

3.6.1. Sequences of Actions and their Properties

Let $P = (FL, \Delta, A, G, N)$ be a normative practical reasoning problem. Also let $\pi = \langle (a_0, 0), \dots, (a_n, t_{a_n}) \rangle$ with $a_i \in A$ and $t_{a_i} \in \mathbb{Z}^+$ be a sequence of actions a_i executed at time t_{a_i} . The pair (a_i, t_{a_i}) reads as action a_i is executed at time $t_{a_i} \in \mathbb{Z}^+$ s.t. $\forall i < j, t_{a_i} < t_{a_j}$. The total duration of a sequence of actions, $Makespan(\pi)$, is defined in Equation 1.

$$Makespan(\pi) = \max(t_{a_i} + d_{a_i}) \quad (1)$$

Actions in a sequence can be executed concurrently but they cannot start at the same time. This is because the planning problem is defined for a single agent and a single agent is not typically assumed to be able to start two actions at the exact same instant. Also actions in the sequence should not have concurrency conflicts, which are defined in Definition 6.

In our presentation we need to check for the occurrence of specific pairs in a sequence of actions π , and we thus define the operator “ $\widehat{\in}$ ” as

$$(a, t_a) \widehat{\in} \pi \text{ iff } \begin{cases} \pi = \langle (a, t_a), \dots, (a_n, t_n) \rangle \text{ or} \\ \pi = \langle (a_0, 0), \dots, (a, t_a), \dots, (a_n, t_n) \rangle \text{ or} \\ \pi = \langle (a_0, 0), \dots, (a, t_a) \rangle \end{cases}$$

Temporary conflicts prevent the agent from executing two actions under specific constraints, the most common one of which is time. Conflicts caused by time, known as *concurrency conflicts* between actions, prevent actions from being executed in an overlapping period of time. We capture in Definition 6 the conditions laid out by Blum and Furst (1997) to establish when two actions cannot be executed concurrently:

Definition 6 (Conflicting Actions). *Actions a_i and a_j have a concurrency conflict if the pre- or post-conditions of a_i contradict the pre- or post-conditions of a_j . The set of conflicting actions is denoted as cf_{action} :*

$$cf_{action} = \left\{ (a_i, a_j) \left| \begin{array}{l} \exists r \in pr(a_i) \cup ps(a_i)^+, \neg r \in pr(a_j) \cup ps(a_j)^- \\ \text{or} \\ \exists \neg r \in pr(a_i) \cup ps(a_i)^-, r \in pr(a_j) \cup ps(a_j)^+ \end{array} \right. \right\} \quad (2)$$

Definition 7 (Sequence of States). *Let $\pi = \langle (a_0, 0), \dots, (a_n, t_{a_n}) \rangle$ be a sequence of actions such that $\nexists (a_i, t_{a_i}), (a_j, t_{a_j}) \in \pi$ s.t. $t_{a_i} \leq t_{a_j} < t_{a_i} + d_{a_i}$, $(a_i, a_j) \in cf_{action}$ and let $m = \text{Makespan}(\pi)$. The execution of a sequence of actions π from a given starting state $s_0 = \Delta$ brings about a sequence of states $S(\pi) = \langle s_0, \dots, s_m \rangle$ for every discrete time interval from 0 to m .*

The transition relation between states is given by Def. 8. If action a_i ends at time k , state s_k results from removing delete post-conditions and adding add post-conditions of action a_i to state s_{k-1} . If there is no action ending at s_k , the state s_k remains the same as s_{k-1} . We first define A_k as the set of action/time pairs such that the actions end at some specific state s_k :

$$A_k = \{(a_i, t_{a_i}) \in \pi \mid k = t_{a_i} + d_{a_i}\} \quad (3)$$

Note that s_k is always well-defined since two actions with inconsistent post-conditions, according to Def. 6 belong to cf_{action} so they cannot be executed concurrently and thus they cannot end at the same state.

Definition 8 (State Transition). *Let $\pi = \langle (a_0, 0), \dots, (a_n, t_{a_n}) \rangle$ and let $S(\pi) = \langle s_0, \dots, s_m \rangle$ be the sequence of states brought about by π :*

$$\forall k > 0 : s_k = \begin{cases} (s_{k-1} \setminus (\bigcup_{a \in A_k} ps(a)^-) \cup \bigcup_{a \in A_k} ps(a)^+ & A_k \neq \emptyset \\ s_{k-1} & A_k = \emptyset \end{cases} \quad (4)$$

We now turn our attention to the properties of each sequence of actions.

Definition 9 (Goal Satisfaction). *Goal requirements should hold in order to satisfy the goal. A sequence of actions $\pi = \langle (a_0, 0), \dots, (a_n, t_{a_n}) \rangle$ satisfies goal g if there is at least one state $s_k \in S(\pi)$ that satisfies the goal:*

$$\pi \models r(g) \text{ iff } \exists s_k \in S(\pi) \text{ s.t. } s_k \models r(g) \quad (5)$$

The set of goals satisfied by π is denoted as G_π :

$$G_\pi = \{g \mid \pi \models r(g)\} \quad (6)$$

Definition 10 (Activated Norms). *A norm $n = \langle d_o, a_{con}, a_{sub}, dl, c \rangle$ is instantiated in a sequence of actions $\pi = \langle (a_0, 0), \dots, (a_n, t_{a_n}) \rangle$ if its activation condition a_{con} belongs to the sequence of actions. Let N_π be the set of instantiations of various norms in π defined in Equation 7. Note that dl_{ins} is calculated based on Definition 5.*

$$N_\pi = \{\langle d_o, a_{con}, a_{sub}, dl_{ins}, c \rangle \mid \langle d_o, a_{con}, a_{sub}, dl, c \rangle \in N, (a_{con}, t_{a_{con}}) \hat{=}\pi\} \quad (7)$$

Definition 11 (Obligation Compliance). A sequence of actions $\pi = \langle (a_0, 0), \dots, (a_n, t_{a_n}) \rangle$ complies with an obligation $n = \langle o, a_{con}, a_{sub}, dl_{ins}, c \rangle$ if a_{con} is executed in π and a_{sub} , starts (cf. Eq. 8) or starts and ends (cf. Eq. 9) within the period when the condition holds and when the deadline expires:

$$\pi \models n \text{ iff } (a_{con}, t_{a_{con}}), (a_{sub}, t_{a_{sub}}) \hat{=}\pi \text{ s.t. } t_{a_{sub}} \in [t_{a_{con}} + d_{a_{con}}, dl_{ins}] \quad (8)$$

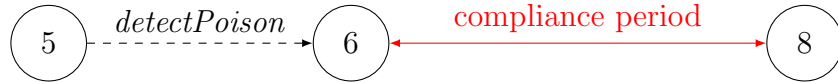
$$\pi \models n \text{ iff } (a_{con}, t_{a_{con}}), (a_{sub}, t_{a_{sub}}) \hat{=}\pi \text{ s.t. } [t_{a_{sub}}, t_{a_{sub}} + d_{a_{sub}}] \subseteq [t_{a_{con}} + d_{a_{con}}, dl_{ins}] \quad (9)$$

Definition 12 (Obligation Violation). A sequence of actions $\pi = \langle (a_0, 0), \dots, (a_n, t_{a_n}) \rangle$ violates obligation $n_{ins} = \langle o, a_{con}, a_{sub}, dl_{ins}, c \rangle$ if a_{con} is executed in π , but a_{sub} does not start (Equation 10), or does not start and end (Equation 11) in the period between the state when the condition holds and when the deadline expires.

$$\pi \not\models n \text{ iff } (a_{con}, t_{a_{con}}) \hat{=}\pi, (a_{sub}, t_{a_{sub}}) \not\hat{=}\pi \text{ s.t. } t_{a_{sub}} \in [t_{a_{con}} + d_{a_{con}}, dl_{ins}] \quad (10)$$

$$\pi \not\models n \text{ iff } (a_{con}, t_{a_{con}}) \hat{=}\pi, (a_{sub}, t_{a_{sub}}) \not\hat{=}\pi \text{ s.t. } [t_{a_{sub}}, t_{a_{sub}} + d_{a_{sub}}] \subseteq [t_{a_{con}} + d_{a_{con}}, dl_{ins}] \quad (11)$$

Example 5. Let $n'_2 = \langle o, detectPoison, stopWater, 8, 10 \rangle$ be the instantiated version of norm $n_2 = \langle o, detectPoison, stopWater, 2, 10 \rangle$ from Example 3. The compliance period for this norm is displayed in the figure below. According to Def. 11 in its Eq. 8, if $t_{stopWater}$ belongs to this period, this norm instance is complied with; otherwise, according to Def. 12 in its Eq. 10, the norm is violated.



Definition 13 (Prohibition Compliance). A sequence of actions $\pi = \langle (a_0, 0), \dots, (a_n, t_{a_n}) \rangle$ complies with prohibition $n = \langle f, a_{con}, a_{sub}, dl_{ins}, c \rangle$ if a_{con} , is

executed and a_{sub} , does not start (Eq. 12) or does not start and end (Eq. 13) in the period when the condition holds and the deadline expires. Formally:

$$\pi \models n \text{ iff } (a_{con}, t_{a_{con}}) \hat{=}\pi, (a_{sub}, t_{a_{sub}}) \hat{\neq}\pi \text{ s.t. } t_{a_{sub}} \in [t_{a_{con}} + d_{a_{con}}, dl_{ins}) \quad (12)$$

$$\pi \models n \text{ iff } (a_{con}, t_{a_{con}}) \hat{=}\pi, (a_{sub}, t_{a_{sub}}) \hat{\neq}\pi \text{ s.t. } [t_{a_{sub}}, t_{a_{sub}} + d_{a_{sub}}] \subseteq [t_{a_{con}} + d_{a_{con}}, dl_{ins}) \quad (13)$$

Definition 14 (Prohibition Violation). A sequence of actions $\pi = \langle (a_0, 0), \dots, (a_n, t_{a_n}) \rangle$ violates prohibition $n = \langle f, a_{con}, a_{sub}, dl_{ins}, c \rangle$ iff a_{con} , has occurred and a_{sub} starts (Eq. 14) or starts and ends (Eq. 15) in the period between when the condition holds and when the deadline expires. Formally:

$$\pi \not\models n \text{ iff } (a_{con}, t_{a_{con}}), (a_{sub}, t_{a_{sub}}) \hat{=}\pi \text{ s.t. } t_{a_{sub}} \in [t_{a_{con}} + d_{a_{con}}, dl_{ins}) \quad (14)$$

$$\pi \not\models n \text{ iff } (a_{con}, t_{a_{con}}), (a_{sub}, t_{a_{sub}}) \hat{=}\pi \text{ s.t. } [t_{a_{sub}}, t_{a_{sub}} + d_{a_{sub}}] \subseteq [t_{a_{con}} + d_{a_{con}}, dl_{ins}) \quad (15)$$

Example 6. Let $n'_4 = \langle f, detectEarthquake, blockMainRoad, 7, 12 \rangle$ be an instantiated version of norm $n_4 = \langle f, detectEarthquake, blockMainRoad, 5, 12 \rangle$, which forbids the agent from blocking the main road within 5 units of time after detecting an earthquake. Since the postconditions of action *blockMainRoad* are brought about at the end of its execution, according to Def. 13 (Eq. 13), this norm is violated if *blockMainRoad* starts and ends between time points 2 and 7. Otherwise, according to Def. 14 (Eq. 15) this norm is complied with.

The set of norms complied with and violated in π are denoted as $N_{cmp(\pi)}$ and $N_{vol(\pi)}$ respectively, and defined as follows:

$$N_{cmp(\pi)} = \{n_{ins} \in N_\pi \mid \pi \models n_{ins}\} \quad (16)$$

$$N_{vol(\pi)} = \{n_{ins} \in N_\pi \mid \pi \not\models n_{ins}\} \quad (17)$$

To make sure there are no norms pending at $m = Makespan(\pi)$, we assume that the norm deadlines are smaller than m . Therefore, all the activated norms in π are either complied with or violated by time m :

$$N_\pi = N_{cmp(\pi)} \cup N_{vol(\pi)} \quad (18)$$

Having defined sequences of actions and their properties, we can now define which sequences of actions can be identified as plans in the next section.

3.6.2. Plans

In classical planning a sequence of actions $\pi = \langle (a_0, 0), \dots, (a_n, t_n) \rangle$ is identified as a plan if all the fluents in the initial state, do hold at time 0 and for each i , the preconditions of action a_i hold at time t_{a_i} , and the goal of planning problem is satisfied in time m , where $m = \text{Makespan}(\pi)$. However, extending the conventional planning problem by multiple potentially conflicting goals and norms requires defining extra conditions in order to make a sequence of actions a plan and a solution for P . In what follows, we define what is required to identify a sequence of actions as a plan.

Definition 15 (Plan). *A sequence of actions $\pi = \langle (a_0, 0), \dots, (a_n, t_n) \rangle$ s.t. $\nexists (a_i, t_{a_i}), (a_j, t_{a_j}) \in \pi$ s.t. $t_{a_i} \leq t_{a_j} < t_{a_i} + d_{a_i}, (a_i, a_j) \in cf_{action}$ is a plan for the normative practical reasoning problem $P = (FL, \Delta, A, G, N)$ if the following conditions hold:*

- *fluents in Δ (and only those fluents) hold in the initial state: $s_0 = \Delta$,*
- *the preconditions of action a_i holds at time t_{a_i} and throughout the execution of a_i : $\forall k \in [t_{a_i}, t_{a_i} + d_{a_i}), s_k \models pr(a_i)$, and*
- *plan π satisfies a non-empty subset of goals: $G_\pi \neq \emptyset$.*

The utility of a plan π is defined by deducting the penalty costs of violated norms from the value gain of satisfying goals (Equation 19). The set of optimal plans, Opt , are those plans that maximise the utility.

$$Utility(\pi) = \sum_{g_i \in G_\pi} v(g_i) - \sum_{n_j \in N_{vol}(\pi)} c(n_j) \quad (19)$$

Examples of calculating the utility of plans are in Appendix B.

The set Opt is empty only if there are no plans for the planning problem. Otherwise, the utility function is guaranteed to terminate and find the optimal plans and hence populate the set Opt .

4. Implementation

In this section, we demonstrate how a normative practical reasoning problem $P = (FL, \Delta, A, G, N)$ (cf. Def. 1), can be implemented. Our implementation should be seen as a proof of concept that provides a computational realisation of all aspects of the formal model. We use Answer Set Programming (ASP) (Gelfond and Lifschitz, 1988) to propose such an implementation. Recent work on planning in ASP (To et al., 2015; Lee and Palla, 2012, 2014) demonstrates that in terms of general planners ASP is a viable competitor.

In what follows, we provide a brief introduction to ASP in Section 4.1, followed by the mapping of normative practical reasoning problem $P = (FL, \Delta, A, G, N)$ (cf. Def. 1) into ASP in Section 4.2. In the latter section we show how P is mapped into an answer set program such that there is a one to one correspondence between solutions for the problem and the answer sets of the program. The mapping itself is provided in Figure 2. The explanation of the mapping is presented in Sections 4.2.1–4.2.6, with cross references to the code fragments listed in Figure 2.

4.1. Answer Set Programming

ASP is a declarative programming paradigm using logic programs under Answer Set semantics (Lifschitz, 2008). Like all declarative paradigms it has the advantage of describing the constraints and the solutions rather than the writing of an algorithm to find solutions to a problem. A variety of programming languages for ASP exist, and we use *AnsProlog* (Baral, 2003). There are several efficient solvers for *AnsProlog*, of which CLINGO (Gebser et al., 2011) and DLV (Eiter et al., 1999) are currently the most widely used.

The basic components of *AnsProlog* are *atoms* that are constructs to which one can assign a truth value. An atom can be negated, adopting *negation as failure* (naf), which establishes that a negated atom not a is true if there is no evidence to prove a . *Literals* are atoms a or negated atoms not a (referred to as naf-literals). Atoms and literals are used to create rules of the general form “ $a :- b_1, \dots, b_m, \text{not } c_1, \dots, \text{not } c_n.$ ” where a, b_i and c_j are atoms. Intuitively, a rule means that *if all atoms b_i are known/true and no atom c_j is known/true, then a must be known/true*. We refer to a as the *head of the rule* and $b_1, \dots, b_m, \text{not } c_1, \dots, \text{not } c_n$ as the *body of the rule*. A rule with an empty body is called a *fact* and a rule with an empty head is called a *constraint*, indicating that no solution should be able to satisfy the body. Another type of rules are called choice rules and are denoted as $l\{h_0, \dots, h_k\}u : - l_1, \dots, l_m, \text{not } l_{m+1}, \dots, \text{not } l_n.$, in which h_i s and l_i s are atoms. l and u are integers and the default values for them are 0 and 1, respectively. A choice rule is satisfied if the number of atoms belonging to $\{h_0, \dots, h_k\}$ that are true/known is between the lower bound l and upper bound u . A *program* is a set of rules representing their conjunction. The semantics of *AnsProlog* is defined in terms of *answer sets*, i.e. assignments of true and false to all atoms in the program that satisfy the rules in a minimal and consistent fashion. A program may have zero or more answer sets, each

corresponding to a solution. We refer to Baral (2003) for a formal treatment of the semantics of ASP.

4.2. Translating the Normative Practical Reasoning Problem into ASP

The description of the mapping of normative practical reasoning problem $P = (FL, \Delta, A, G, N)$ into ASP is presented in the following sections with references to Figure 2. Note that variables are in capitals and grounded variables are in small italics.

4.2.1. States

In Section 3.6 we described the semantics $P = (FL, \Delta, A, G, N)$ over a set of states. The facts produced by line 1 provide the program with all available states for plans of maximum length q ³. Line 2 encodes that the initial fluents, $(x \in \Delta)$ need to hold at state 0 which is achieved by the facts `holdsat(x,0)`. Fluents are inertial, they continue to hold unless they are terminated. Inertia is encoded in lines 3–4. Termination is modelled through the predicate `terminated(X,S)`.

4.2.2. Actions

Each durative action is encoded as `action(a,d)` (line 5), where *a* is the name of the action and *d* is its duration. The preconditions $pr(a)$ of action *a* hold in state *s* if $s \models pr(a)$. This is expressed in line 6 using atom `pre(a,S)`. In order to make the coding more readable we introduce the shorthand `EX(X,S)`, where *X* is a set of fluents that should hold at state *S*. For all $x \in X$, `EX(X,S)` is translated into `holdsat(x,S)` and for all $\neg x \in X$, `EX($\neg X$,S)` is translated into `not EX(x,S)` using negation as failure.

The agent has the choice to execute any of its actions in any state. This is expressed in the choice rule in line 7. Since no lower or upper bound is given for `{executed(A,S)}`, the default value of `0{executed(A,S)}1` is implied, meaning that the agent has the choice of whether or not to execute an action. Following the approach in Blum and Furst (1997), we assume that the preconditions of a durative action should be preserved when it is in progress. We first encode the description of an action in progress, followed by ruling out the possibility of an action being in progress in the absence of its preconditions. A durative action is in progress, `inprog(A,S)`, from the

³Currently, the length of the plans needs to be determined experimentally. We plan to automate this using incremental features of ASP solver CLINGO4 (Gebser et al., 2011).

state in which it starts up to the state in which it ends (lines 8–9). Line 10, rules out the execution of an action, when the preconditions of the action do not hold during its execution. A further assumption made is that the agent cannot start two actions at exactly the same time (line 11–12). Once an action starts, the result of its execution is reflected in the state where the action ends. This is expressed through (i) lines 13–14 that allow the add postconditions of the action to hold when the action ends, and (ii) line 15–16 that allow the termination of the delete postconditions. Termination takes place in the state *before* the end state of the action: the reason for this is the inertia of fluents that was expressed in lines 3–4. Thus delete post-conditions of an action are terminated in the state before the end state of the action, so that they will not hold in the following state, in which the action ends (i.e. they are deleted from the state).

4.2.3. Goals and Norms

Line 17 encodes goal g with value v as a fact. Goal g is satisfied in state s if $s \models g$. This is expressed in line 18, where g^+ and g^- are the positive and negative literals in the set g .

For the norms we note that, following Definitions 11–14, compliance and violation of a norm can be established based on the start state of action’s execution that is the subject of the norm, or at the end state of action’s execution. In the encoding we show an implementation of the former; the latter can be catered for in a similar fashion.

Lines 19–39 deal with obligations and prohibitions of the form $n = \langle d_o, a_{con}, a_{sub}, dl, c \rangle$. Line 19 encodes norm n with penalty cost c upon violation. In order to implement the concepts of norm compliance and violation for instantiated norms, we introduce a normative fluent that holds over the compliance period. The compliance period begins from the state in which action a_{con} ’s execution ends. The compliance period then ends within dl time units of the end of action a_{con} , which is denoted as dl' in the normative fluent. For instance, fluent $o(n_1, s', a_{sub}, dl')$ expresses that the instance of norm n_1 that was activated in state s' , obliges the agent to execute action a_{sub} before deadline dl' . The state in which the norm is activated is a part of the fluent to distinguish different activations of the same norm from one another. For example, fluent $o(n_1, s'', a_{sub}, dl'')$ refers to a different instance of norm n_1 that was activated in s'' . An obligation fluent denotes that action a_{sub} ’s execution should begin before deadline dl' or be subject to violation, while prohibition fluent $f(n_2, s', a_{sub}, dl')$ denotes that action a_{sub} should not

Creating states: $\forall k \in [0, q]$

```
1 state(k).
```

Setting up the initial state: $\forall x \in \Delta$

```
2 holdsat(x,0).
```

Rule for fluent inertia

```
3 holdsat(X,S2) :- holdsat(X,S1), not terminated(X,S1),  
4                  state(S1), state(S2), S2=S1+1.
```

Creating the actions and their preconditions: $\forall a \in A, a = \langle pr, ps, d \rangle$

```
5 action(a,d).  
6 pre(a,S) :- EX(pr(a)+,S), not EX(pr(a)-,S), state(S).
```

Common constraints on action execution

```
7 {executed(A,S)} :- action(A,D), state(S).  
8 inprog(A,S2) :- executed(A,S1), action(A,D),  
9                state(S1), state(S2), S1<=S2, S2<S1+D.  
10 :- inprog(A,S), action(A,D), state(S), not pre(A,S).  
11 :- executed(A1,S), executed(A2,S), A1!=A2,  
12    action(A1,D1), action(A2,D2), state(S).
```

Adding positive postconditions of actions: $ps(a)^+ = X \Leftrightarrow \forall x \in X$.

```
13 holdsat(x,S2) :- executed(a,S1), action(a,d),  
14                state(S1), state(S2), S2=S1+d.
```

Terminating negative post conditions of actions: $ps(a)^- = X \Leftrightarrow \forall x \in X$.

```
15 terminated(x,S2) :- executed(a,S1), action(a,d),  
16                  state(S1), state(S2), S2=S1+d-1.
```

Creating the goals: $\forall g \in G$

```
17 goal(g,v).  
18 satisfied(g,S) :- EX(g+,S), not EX(g-,S), state(S).
```

Creating the norms: $\forall n = \langle o|f, a_{sub}, a_{con}, dl, c \rangle \in N$

19 `norm(n, c).`

$\forall n = \langle o, a_{sub}, a_{con}, dl, c \rangle \in N$

```

20 holdsat(o(n, S1, a_sub, dl+S2), S2) :- executed(a_con, S1),
21      action(a_con, d), S2=S1+d, state(S1), state(S2).
22 cmp(o(n, S1, a, DL), S2) :- holdsat(o(n, S1, a, DL), S2),
23      executed(a, S2), action(a, d), state(S1), state(S2), S2!=DL.
24 terminated(o(n, S1, a, DL), S2) :- cmp(o(n, S1, a, DL), S2),
25      state(S1), state(S2).
26 vol(o(n, S1, a, DL), S2) :- holdsat(o(n, S1, a, DL), S2), DL=S2,
27      state(S1), state(S2).
28 terminated(o(n, S1, a, DL), S2) :- vol(o(n, S1, a, DL), S2),
29      state(S1), state(S2).

```

$\forall n = \langle f, a_{sub}, a_{con}, dl, c \rangle \in N$

```

30 holdsat(f(n, S1, a_sub, dl+S2), S2) :- executed(a_con, S1),
31      action(a_con, d), S2=S1+d, state(S1), state(S2).
32 cmp(f(n, S1, a, DL), S2) :- holdsat(f(n, S1, a, DL), S2),
33      action(a, d), DL=S2, state(S1), state(S2).
34 terminated(f(n, S1, a, DL), S2) :- cmp(f(n, S1, a, DL), S2),
35      state(S1), state(S2).
36 vol(f(n, S1, a, DL), S) :- holdsat(f(n, S1, a, DL), S2),
37      executed(a, S2), state(S1) state(S2), S2!=DL.
38 terminated(f(n, S1, a, DL), S2) :- vol(f(n, S1, a, DL), S2),
39      state(S1), state(S2).

```

Plans need to satisfy at least one goal

```

40 satisfied(g) :- satisfied(g, S), state(S).
41 :- not satisfied(g1), ... , not satisfied(gm).

```

Avoiding conflicting actions: $\forall (a_1, a_2) \in cf_{action}$

```

42 :- inprog(a1, S), inprog(a2, S), action(a1, d1), action(a2, d2),
43      state(S).

```

Figure 2: Mapping $P = (FL, I, A, G, N)$ to its Corresponding Computational Model Π_{PBase}

begin before deadline dl' or be subject to violation. Lines 20–21 and 30–31 establish respectively the obligation and prohibition fluents that hold for the duration of the compliance period.

In terms of compliance, if the obliged action *starts* during the compliance period in which the obligation fluent holds, the obligation is complied with (line 22–23). Compliance is denoted by the atom `cmp`. The obligation fluent is terminated in the same state that compliance is detected (lines 24–25). If the deadline expires and the obligation fluent still holds, it means that the compliance never occurred during the compliance period and the norm is therefore violated (lines 26–27). The atom `vol` denotes violation. The obligation fluent is terminated when the deadline expires and the norm is violated (lines 28–29).

On the other hand, a prohibition norm is violated if the forbidden action *begins* during the compliance period in which the prohibition fluent holds (lines 36–37). As with the obligation norms, after being violated, the prohibition fluent is terminated (lines 38–39). If the deadline expires and the prohibition fluent still holds, that means the prohibited action did not begin during the compliance period and the norm is therefore complied with (lines 32–33). The obligation fluent is terminated in the same state that compliance is detected (lines 34–35).

4.2.4. Mapping Answer Sets to Plans

Having implemented the components of $P = (FL, \Delta, A, G, N)$, we now encode the criteria for a sequence of actions to be identified as a plan and a solution to P . The rule in line 41 is responsible for constraining the answer sets to those that fulfill at least one goal. This is done by excluding answers that do not satisfy any goal. The input for this rule is provided in line 40, where goals are marked as satisfied if they are satisfied in at least one state. Prevention of concurrent conflicting actions is achieved via lines 42–43 which establish that two such actions cannot be in progress simultaneously. This concludes the mapping of a formal planning problem to its computational counterpart in *AnsProlog*. For a problem P we refer to the program consisting of lines 1–43 as Π_{PBase} .

4.2.5. Soundness and Completeness of Implementation

The following theorems state the correspondence between the solutions for problem P and answer sets of program Π_{PBase} .

Theorem 1 (Soundness). *Let $P = (FL, I, A, G, N)$ be a normative practical reasoning problem with Π_{PBase} as its corresponding AnsProlog program. Let Ans be an answer set of Π_{PBase} , then a set of atoms of the form $executed(a_i, t_{a_i}) \in Ans$ encodes a solution to P .*

Due to space limitation, we are not able to present the proof of this theorem here. However, the proof is available in a technical report (Shams et al., 2017). This is a proof by structure that explains how the structure of Π_{PBase} satisfies the conditions that identifies a sequence of actions as a plan.

Theorem 2 (Completeness). *Let $\pi = \langle (a_0, 0), \dots, (a_n, t_{a_n}) \rangle$ be a plan for $P = (FL, I, A, G, N)$. Then there exists an answer set of Π_{PBase} containing atoms $executed(a_i, t_{a_i})$ that correspond to π .*

The proof of this theorem is also presented in (Shams et al., 2017). In this proof the program is first transformed to a program without any naf-literals and choice rules. We then take a candidate answer set for the program and show that it is a minimal model for the transformed program.

4.2.6. Optimal Plans

In order to find optimal plans in Figure 3 we show how to encode the utility function defined by Eq. 19. The sum of values of goals satisfied in a plan is calculated in line 44, where we use an ASP built-in aggregate `#sum`. This aggregate is an operation on a set of weighted literals that evaluates to the sum of the weights of the literals. We first assign the value of goals as the weight of literals `satisfied(G)` and then use `#sum` to compute the sum of value of all goals satisfied.

The sum of costs of norms violated in a plan is calculated in line 47 using the same aggregate. However, the weight of the literal is the cost of punishment of the norms. The input for this line is given in lines 45 and 46, where violated norms are distinguished based on the combination of the norm id n and the state s in which they are instantiated. Having calculated `value(TV)` and `cost(TC)`, the utility of a plan is computed in line 48, which is subject to a built-in optimisation statement in the final line. This optimisation statement identifies an answer set as optimal if the sum of weights of literals that hold is maximal with respect to all answer sets of the program. By assigning U as the weight of literal `utility(U)` we compute the answer set that maximises the utility.

```

44 value(TV) :- TV = #sum {V: goal(G,V), satisfied(G)}.
45 violated(N,S1) :- vol(o(N,S1,a,DL),S2), state(S1;S2).
46 violated(N,S1) :- vol(f(N,S1,a,DL),S2), state(S1;S2).
47 cost(TC) :- TC = #sum{C,S:violated(N,S),norm(N,C)}.
48 utility(TV-TC) :- value(TV), cost(TC).
49 #maximize {U:utility(U)}.

```

Figure 3: Optimal Plans for $P = (FL, I, A, G, N)$

Let program $\Pi_P = \Pi_{PBase} \cup \Pi_P^*$, where Π_P^* consists of lines 44–49. The following theorem states the correspondence between the plans for problem P and answer sets of program Π_P .

Theorem 3. *Given a normative practical reasoning problem $P = (FL, I, A, G, N)$, for each answer set Ans of Π the set of atoms of the form $executed(a_i, t_{a_i})$ in Ans encodes an optimal solution to P . Conversely, each solution to the problem P corresponds to a single answer set of π .*

This theorem follows immediately from Theorem 1 and 2 and the structure of program Π_P^* .

5. Related Work

We consider the approaches to accounting for norms in practical reasoning, with the aim of identifying similarities across a variety of architectures. Much work stems from planning and how to account for norms in the plan construction and selection process. There are broadly three approaches:

1. Choosing a plan that is norm compliant (e.g., NoA (Kollingbaum, 2005)), which is a one-off process, that may fail delivering the best (where “best” can be defined in various ways) plan available for the situation from those available, and which requires starting again when a plan step fails and the remainder of the plan is invalidated. The main points of difference between NoA and the work presented here are that (i) NoA agents are BDI specific, (ii) they do not have internal motivations such as goals or values that might conflict with norms, which therefore enables the NoA agent to always comply with norms (iii) plans are pre-existing rather than generated.

Table 1: Summary of Related Frameworks

Framework	Deontic Operator	Activation Condition	De-activation Condition
BOID (Broersen et al., 2001)	o	N/A	N/A
NoA (Kollingbaum, 2005)	o, f, p	state, action	state, action
ν -BDI (Meneguzzi et al., 2015)	o, f	state	state
N-2APL (Alechina et al., 2012)	o, f	state	state ⁴
N-Jason (Lee et al., 2014)	o, p, w ⁵	N/A	temporal constraint
Oren et al. (2011)	o, f	N/A	N/A
Panagiotidi et al. (2012a)	o, f	state	state
Shams et al. (2015)	o, f	action	temporal constraint
This work	o, f	action	temporal constraint

2. Customising a plan to make it norm compliant (e.g., Oren et al. (2011)) is potentially more flexible in making use of available plans (also helping customize existing plans into optimal norm-compliant plans), but otherwise has the same replanning drawback when a plan step fails. In common with Oren et al. (2011), we use the utility of the entire plan in the selection process, but differ in that we generate plans rather than use a plan library.
3. Generating a plan that is norm compliant (e.g., Panagiotidi et al. (2012a); Shams et al. (2015)). The former addresses on-going compliance and re-planning, putting a high computational overhead on each plan step. Of necessity, Panagiotidi et al. (2012a) can only compute utility on a step-by-step basis, whereas in this work consider the utility of the whole plan. Shams et al. (2015) attempt to balance compliance on the part of the agent (where the agent chooses a norm-compliant action in preference) with enforcement (where the agent is discouraged from non-norm-compliance via punishments for norm violation), but is not robust to plan failure. Furthermore, in (Shams et al., 2015), conflict is formulated in advance by taking a static view about conflicts. For instance, two goals that are logically inconsistent, cannot be satisfied in the same plan, regardless of the order or choice of actions in a plan. In contrast, in the work presented here, conflicts are not formulated in advance; instead, they can be inferred from plans. Therefore, the agent might be able to schedule its actions such that two goals that are

⁴The de-activation condition only applies to obligations.

⁵The operator w denotes power and indicates the capability to do something in prohibitive societies, where actions are not allowed unless empowered and explicitly permitted.

logically inconsistent are satisfied in the same plan at different points in time. Finally, as discussed in Section 3.6, we extend the norm representation of (Shams et al., 2015) to accommodate compliance and violation in the presence of durative actions more flexibly by allowing compliance to be defined as the start or end of the action that is the subject of the norm.

Table 1 provides a feature analysis of various related frameworks and that proposed here. In addition to those discussed in some detail above, we highlight: (i) BOID (Broersen et al., 2001) which is very expressive, but of historical interest, since no reference implementation is currently available (ii) ν -BDI (Meneguzzi et al., 2015) which draws on BOID and NoA, but emphasizes practicality and computational efficiency (iii) N-2APL (Alechina et al., 2012), in which norms are either obligations or prohibitions and plans are not interleaved, and (iv) N-Jason (Lee et al., 2014) that allows agents to reason about norms deadlines and priorities, and choose between plans triggered by a particular norm.

The majority of frameworks, including ours, deal with obligations and prohibitions. The activation condition, however, varies with those that support conditional norms. An activation condition presented as an action can be expressed as a state that satisfies the post-conditions of the action. Unlike the other frameworks, ours exploits the explicit representation of time in the formal model to encode the norm de-activation condition as a time instant. Associating a deadline with temporal properties is considered to be realistic and dynamic, in particular when the norms capture the requirements of real-world scenarios (Chesani et al., 2013; Kafali et al., 2014; Gasparini et al., 2015), such as the disaster scenario modelled in this paper. A further key differentiator between our work and the others discussed here, is the capacity to handle durative actions and their concurrent execution, as well as dealing with norm compliance and violation in the presence of durative actions.

6. Conclusions and Future Work

An agent performing practical reasoning in an environment regulated by norms constantly needs to weigh up the importance of goals to be satisfied and norms with which to comply. This decision process is only possible when the agent has access to the set of all possible plans available and the agent can ascertain the impact of its decision on entire plans. This research offers means to capture and measure the impact via utility functions, offering numeric

metrics, so that the decision problem can be reformulated as choosing a plan from a set of generated plans, which maximises its overall utility. While the literature we have surveyed contains practical reasoning frameworks that take into account normative considerations, they are limited in several ways, and we have contrasted them with our approach in the previous section.

The majority of these frameworks are limited to a specific type of agent architecture, mostly BDI, (e.g., Kollingbaum (2005); Meneguzzi et al. (2015)). We do not assume any specific architecture. Instead, we adopt a realistic view that agents have capabilities (encoded as the actions they may perform), and they have internal motivations (encoded as the goals of a planning problem). This leaves the option of extending current work to a multi-agent setting where agents might not necessarily have the same architecture.

In the approaches set out in the literature the attitude agents have towards norms is often one of compliance, meaning that their plans are often selected or, in some approaches, customised, to ensure norm compliance, (e.g., Kollingbaum (2005); Alechina et al. (2012); Oren et al. (2011)). We argue that in certain situations an agent might be better off violating a norm which, if followed, would make it impossible for the agent to achieve an important goal or complying with a more important norm; we enable agents to compare the relative importance of goals and norms via their utilities. Consequently, through our practical reasoning process agents consider *all* plans (i.e., sequences of actions), including those leading to norm compliance and violation; each plan gets an associated overall utility for its sequences of actions, and norms followed/violated, and agents can decide which of them to pursue. The plan an agent chooses to follow is not necessarily norm-compliant, however, our mechanism guarantees that the decision will maximise the overall plan/norm utility, which justifies the occasional violation of norms as the plan is followed.

We see several interesting avenues for future work. Our research currently addresses normative practical reasoning in a single-agent setting, extending to a multi-agent setting seems a natural next step. This idea can be explored both when the agents are collaborating to fulfill common goals, as well as when they are competing to use resources to fulfill their individual goals. In the former case, the best course of action can be identified as one that maximises the overall utility of the system. In the latter, game-theoretic approaches can be utilised to identify a solution that ensure relative optimality for the individuals (e.g. Ågotnes et al. (2007)).

We note the relative limitations of our norm representation. Although our approach addressed action-based norms, we envisage how it can be extended and adapted to handle state-based norms. Our Def. 4 needs to cater for formulae to represent both the norm activation condition, a_{con} , and the norm subject, a_{sub} , instead of actions. A combination of action- and state-based norms (e.g. De Vos et al. (2013)) enriches the norm representation as well as normative reasoning. Also, the norm representation language can be extended to cater for deadlines that are expressed as reaching a state – In such cases the deadline is referred to as a norm termination condition – rather than a time instance. For instance, an obligation to open a dam on a river can come in force when the water level is above a certain point, and subsequently terminated when the water level drops below a certain level, regardless of how long it takes for that to happen. We would also like to include permission norms in addition to obligations and prohibitions. The modelling of permissions as exceptions to obligations and prohibitions has been used to justify violations under specific circumstances, (e.g. Oren et al. (2010); Pacheco (2012)). Last interesting point to consider is to factor in variant penalty cost for norm violation. For instance, if an agent violates the same norm repeatedly the penalty cost can be increased to enforce the agent to devise another plan.

Finally, our implementation should be seen as a proof-of-concept that, apart from replanning, provides a provable computational realisation of all aspects of the formal model. In future, we aim at extending the implementation to accommodate replanning when a plan in progress is interrupted for any reason. The formal model is implementation language neutral so other implementation languages could be used.

Appendix A. Formulation of the Disaster Scenario

We provide a formalisation of the scenario set out in Section 2. Let $P = (FL, \Delta, A, G, N)$ be the normative practical reasoning problem for the disaster scenario such that:

$$\bullet FL = \left\{ \begin{array}{l} shockDetected, poisonDetected, waterSupplied, \\ areaSecured, evacuated, shockDetected, \\ shelterBuilt, populated, wounded, \\ earthquakeDetected, medicineSupplied, \\ noAccess, medicsPresent \end{array} \right\}$$

- $\Delta = \left\{ \begin{array}{l} earthquakeDetected, medicsPresent, \\ wounded, populated, waterSupplied \end{array} \right\}$
- $A = \left\{ \begin{array}{l} detectShock, detectPoison, stopWater, \\ buildShelter, evacuate, getMedicine, secure \end{array} \right\}$ where
 $detectShock = \langle \{\}, \{shockDetected\}, 1 \rangle$.
 $detectPoison = \langle \{\}, \{poisonDetected\}, 1 \rangle$.
 $stopWater = \left\langle \left\{ \begin{array}{l} poisonDetected, \\ waterSupplied \end{array} \right\}, \{\neg waterSupplied\}, 1 \right\rangle$
 $buildShelter = \left\langle \left\{ \begin{array}{l} areaSecured, \\ evacuated, \\ \neg shockDetected \end{array} \right\}, \left\{ \begin{array}{l} shelterBuilt, \\ \neg evacuated \end{array} \right\}, 4 \right\rangle$
 $evacuate = \left\langle \left\{ \begin{array}{l} shockDetected, \\ populated \end{array} \right\}, \left\{ \begin{array}{l} evacuated, \\ \neg populated \end{array} \right\}, 5 \right\rangle$
 $getMedicine = \left\langle \left\{ \begin{array}{l} earthquakeDetected, \\ wounded \end{array} \right\}, \{medicine\}, 3 \right\rangle$
 $secure = \left\langle \{evacuated\}, \left\{ \begin{array}{l} areaSecured, \\ noAccess \end{array} \right\}, 3 \right\rangle$
- $G = \{runningHospital, organiseSurvivorCamp\}$, where:
 $runningHospital = \left\langle \left\{ \begin{array}{l} medicsPresent, \\ waterSupplied, \\ medicineSupplied \end{array} \right\}, 25 \right\rangle$
 $organiseSurvivorCamp = \left\langle \left\{ \begin{array}{l} areaSecured, \\ shelterBuilt \end{array} \right\}, 18 \right\rangle$
- $N = \{n_1, n_2\}$, where:
 $n_1 = \langle f, detectShock, buildShelter, 3, 5 \rangle$
 $n_2 = \langle o, detectPoison, stopWater, 2, 10 \rangle$

Appendix B. Mapping of Our Disaster Scenario

The formal specification of our disaster scenario (Section2) as provided in the previous section can be mapped to its corresponding *AnsProlog* program following the rules given in Figure 2 on page 20. Optimisation conditions are

similarly mapped to their corresponding *AnsProlog* program following the rules given in Figure 3 on page 23. We refer the readers to (Shams et al., 2017) for the *AnsProlog* program of this scenario. The graphic representation of the three answer sets of the program is given in Figs. B.4–B.6, where arcs show the actions in progress and the boxes below each state, show the fluents that hold in that state. The fluents in bold are the fluents that are added to the state, while the crossed fluents are the terminated ones. Norms violated in a state are highlighted in red and goals satisfied are highlighted in green. Applying the optimisation statements in Fig. 3, the utility of each plan presented by each answer set is calculated as below, making the plan presented by answer set 3 the optimal plan:

- Utility of plan given by answer set 1 shown in Fig. B.4:
 $Utility(\pi_{Ans1}) = v(\textit{runningHospital}) = 25$
- Utility of plan given by answer set 2 shown in Fig. B.5:
 $Utility(\pi_{Ans2}) = v(\textit{runningHospital}) + v(\textit{organiseSurvivorCamp}) - c(n_1) - c(n_1) = 25 + 18 - 5 - 5 = 33$
- Utility of plan given by answer set 3 shown in Fig. B.6:
 $Utility(\pi_{Ans3}) = v(\textit{runningHospital}) + v(\textit{organiseSurvivorCamp}) = 43$







References

- Ågotnes, T., van der Hoek, W., Wooldridge, M., 2007. Normative system games. In: Durfee, E. H., Yokoo, M., Huhns, M. N., Shehory, O. (Eds.), International Conference on Autonomous Agents and Multiagent Systems. IFAAMAS, p. 129.
- Aldewereld, H., Dignum, F., García-Camino, A., Noriega, P., Rodríguez-Aguilar, J. A., Sierra, C., 2006. Operationalisation of norms for usage in electronic institutions. In: Nakashima, H., Wellman, M. P., Weiss, G., Stone, P. (Eds.), International Conference on Autonomous Agents and Multiagent Systems. ACM, pp. 223–225.
- Alechina, N., Dastani, M., Logan, B., 2012. Programming norm-aware agents. In: van der Hoek, W., Padgham, L., Conitzer, V., Winikoff, M. (Eds.), International Conference on Autonomous Agents and Multiagent Systems. IFAAMAS, pp. 1057–1064.
- Alrawagfeh, W., Meneguzzi, F., 2014. Utilizing permission norms in BDI practical normative reasoning. In: International Workshop on Coordination, Organizations, Institutions, and Norms. Springer International Publishing, pp. 1–18.
- Baral, C., 2003. Knowledge Representation, Reasoning, and Declarative Problem Solving. Cambridge University Press, New York, NY, USA.
- Blum, A. L., Furst, M. L., 1997. Fast planning through planning graph analysis. *Artificial Intelligence* 90 (1), 281 – 300.
- Broersen, J., Dastani, M., Hulstijn, J., Huang, Z., van der Torre, L., 2001. The BOID architecture: Conflicts between beliefs, obligations, intentions and desires. International Conference on Autonomous Agents. ACM, pp. 9–16.
- Chesani, F., Mello, P., Montali, M., Torroni, P., 2013. Representing and monitoring social commitments using the event calculus. *Autonomous Agents and Multi-Agent Systems* 27 (1), 85–130.
- Cliffe, O., De Vos, M., Padget, J. A., 2006. Answer set programming for representing and reasoning about virtual institutions. In: Inoue, K., Satoh,

- K., Toni, F. (Eds.), *International Workshop on Computational Logic in Multi-Agent Systems*. Springer, pp. 60–79.
- Criado, N., Argente, E., Botti, V. J., 2010. A BDI architecture for normative decision making. In: van der Hoek, W., Kaminka, G. A., Lespérance, Y., Luck, M., Sen, S. (Eds.), *International Conference on Autonomous Agents and Multiagent Systems*. IFAAMAS, pp. 1383–1384.
- De Vos, M., Balke, T., Satoh, K., 2013. Combining event-and state-based norms. In: *International conference on Autonomous Agents and Multi-Agent Systems*. pp. 1157–1158.
- Doherty, P., Gustafsson, J., Karlsson, L., Kvarnström, J., 1998. TAL: temporal action logics language specification and tutorial. *Electronic Transactions on Artificial Intelligence* (2), 273–306.
- Eiter, T., Faber, W., Leone, N., Pfeifer, G., 1999. The diagnosis frontend of the DLV system. *AI Communications* 12, 12–1.
- Esteva, M., Rodríguez-Aguilar, J. A., Sierra, C., Garcia, P., Arcos, J. L., 2001. On the formal specifications of electronic institutions. In: Dignum, F., Sierra, C. (Eds.), *Agent Mediated Electronic Commerce, The European AgentLink Perspective*. Vol. 1991 of *Lecture Notes in Computer Science*. Springer, pp. 126–147.
- Fikes, R. E., Nilsson, N. J., 1971. Strips: A new approach to the application of theorem proving to problem solving. *International Joint Conference on Artificial Intelligence*. Morgan Kaufmann Publishers Inc., pp. 608–620.
- Gasparini, L., Norman, T. J., Kollingbaum, M. J., Chen, L., Meyer, J. C., 2015. Verifying normative system specification containing collective imperatives and deadlines. In: Weiss, G., Yolum, P., Bordini, R. H., Elkind, E. (Eds.), *International Conference on Autonomous Agents and Multiagent Systems*. ACM, pp. 1821–1822.
- Gebser, M., Kaminski, R., Kaufmann, B., Ostrowski, M., Schaub, T., Schneider, M., 2011. Potassco: The Potsdam answer set solving collection. *AI Communications* 24 (2), 107–124.

- Gelfond, M., Lifschitz, V., 1988. The stable model semantics for logic programming. In: Kowalski, R. A., Bowen, K. A. (Eds.), *International Conference and Symposium on Logic Programming*. MIT Press, pp. 1070–1080.
- Gelfond, M., Lifschitz, V., 1998. Action languages. *Electronic Transactions on Artificial Intelligence* 2, 93–210.
- Kafali, Ö., Günay, A., Yolum, P., 2014. GOSU: computing goal support with commitments in multiagent systems. In: Schaub, T., Friedrich, G., O’Sullivan, B. (Eds.), *European Conference on Artificial Intelligence*. Vol. 263 of *Frontiers in Artificial Intelligence and Applications*. IOS Press, pp. 477–482.
- Kollingbaum, M., 2005. Norm-governed practical reasonig agents. Ph.D. thesis, University of Aberdeen.
- Kowalski, R., Sergot, M., Jan. 1986. A logic-based calculus of events. *New Generation Computing*. 4(1), 67–95.
- Lee, J., Padget, J., Logan, B., Dybalova, D., Alechina, N., 2014. N-Jason: Run-time norm compliance in AgentSpeak(L). In: Dalpiaz, F., Dix, J., van Riemsdijk, M. B. (Eds.), *International Workshop in Engineering Multi-Agent Systems*. Vol. 8758 of *Lecture Notes in Computer Science*. Springer, pp. 367–387.
- Lee, J., Palla, R., 2012. Reformulating temporal action logics in answer set programming. In: Hoffmann, J., Selman, B. (Eds.), *Conference on Artificial Intelligence*. AAAI Press, 786–792.
- Lee, J., Palla, R., 2014. Reformulating the situation calculus and the event calculus in the general theory of stable models and in answer set programming. *CoRR* abs/1401.4607.
- Lifschitz, V., 2002. Answer set programming and plan generation. *Artificial Intelligence* 138(1-2), 39–54.
- Lifschitz, V., 2008. What is answer set programming? In: Fox, D., Gomes, C. P. (Eds.), *International Conference on Artificial Intelligence*. AAAI, pp. 1594–1597.

- López y López, F., Luck, M., d’Inverno, M., 2005. A normative framework for agent-based systems. In: Normative Multi-Agent Systems (NORMAS). pp. 24–35.
- Meneguzzi, F., Rodrigues, O., Oren, N., Vasconcelos, W. W., Luck, M., 2015. BDI reasoning with normative considerations. *Engineering Application of Artificial Intelligence* 43, 127–146.
- Nunes, I., Fiadeiro, J. L., Turski, W. M., 1997. Coordination durative actions. In: Garlan, D., Métayer, D. L. (Eds.), *International Conference on Coordination Languages and Models*. Vol. 1282 of *Lecture Notes in Computer Science*. Springer, pp. 115–130.
- Oren, N., Croitoru, M., Miles, S., Luck, M., 2010. Understanding permissions through graphical norms. In: Omicini, A., Sardiña, S., Vasconcelos, W. W. (Eds.), *Declarative Agent Languages and Technologies*. Vol. 6619 of *Lecture Notes in Computer Science*. Springer, pp. 167–184.
- Oren, N., Vasconcelos, W., Meneguzzi, F., Luck, M., 2011. Acting on norm constrained plans. In: Leite, J., Torroni, P., Ågotnes, T., Boella, G., van der Torre, L. (Eds.), *Computational Logic in Multi-Agent Systems*. Vol. 6814 of *Lecture Notes in Computer Science*. Springer, pp. 347–363.
- Pacheco, N. C., 2012. Using norms to control open multi-agent systems. Ph.D. thesis, Universidad Politecnica de Valencia.
- Panagiotidi, S., Vázquez-Salceda, J., Dignum, F., 2012a. Reasoning over norm compliance via planning. In: Aldewereld, H., Sichman, J. S. (Eds.), *International Workshop on Coordination, Organizations, Institutions, and Norms in Agent Systems*. Vol. 7756 of *Lecture Notes in Computer Science*. Springer, pp. 35–52.
- Panagiotidi, S., Vázquez-Salceda, J., Vasconcelos, W., 2012b. Contextual norm-based plan evaluation via answer set programming. In: Pérez, J. B., Sánchez, M. A., Mathieu, P., Rodríguez, J. M. C., Adam, E., Ortega, A., García, M. N. M., Navarro, E., Hirsch, B., Cardoso, H. L., Julián, V. (Eds.), *International Conference on Practical Applications of Agents and Multi-Agent Systems*. Vol. 156 of *Advances in Soft Computing*. Springer, pp. 197–206.

- Pitt, J., Busquets, D., Riveret, R., Sept 2013. Formal models of social processes: The pursuit of computational justice in self-organising multi-agent systems. In: International Conference on Self-Adaptive and Self-Organizing Systems. pp. 269–270.
- Shams, Z., 2016. An Argumentation-Based Approach to Normative Practical Reasoning. Ph.D. thesis, University of Bath.
- Shams, Z., De Vos, M., Padget, J., Vasconcelos, W. 2015. Implementation of normative practical reasoning with durative actions. In: Coordination, Organizations, Institutions, and Norms in Agent Systems. Springer, pp. 136–154.
- Shams, Z., De Vos, M., Padget, J., Vasconcelos, W. W., 2017. Practical reasoning with norms for autonomous software agents (full edition). In: arXiv 1701.08306.
- To, S. T., Son, T. C., Pontelli, E., 2015. A generic approach to planning in the presence of incomplete information: Theory and implementation. *Artificial Intelligence* 227, 1–51.